# ST-Stepper Motor Control Board User's Guide
# Table of Contents


**Chapter 1: Introduction**

Features 1
Quick Start 2

**Chapter 2: Function Description**

Stepper Motor Control 3
- Stepper Motor Control IC 3
- The Driver IC 3

Communication with PCs 3
- ITC232-A 4
- Networking with an I/O Board 4

Other Features 4
- A/D Converter 4
- IRQ, PWM and I/O Ports 4

Software 4

**Chapter 3: Operation of the ST Board**

Connections 5
- Connections of Stepper Motors 5
- Power Supply for the Board 5
- Power Supply for the Stepper Motors 5

Communications with PCs 5
Controlling Stepper Motors 5
- Initializing the Board 5
- Sending Commands to the CY500 ICs 6
- Moving the Stepper Motors 7
- Stopping the Stepper Motors 7
- Triggering the Stepper Motors 8
- Home Buttons 8

A/D Conversion 8
- Reset 8
- Initialization, Getting Conversion Results 8

**Chapter 4: Programming the ST Using the Provided DLL**

Using the DLL with Different Programming Languages 8
Controlling the Motors 9
- Establishing the Connection Between a PC and ST 9
- Initializing the ST 9
- Configuring the Motors 9
- Turning and Stopping the Motors 9
- Motor Homing 9
- Motor Command 10
- Error Message 10

A/D Conversion 10
Network Functions 10

**Appendix A: DLL Functions**

# ST-Stepper Motor Control Board User's Guide

***DISCLAIMER:*** *RMV ELECTRONICS INC. does not assume any liability arising from the application and/or use of the product/s described herein, nor does it convey any license. RMV ELECTRONICS INC. products are not authorized for use as components in medical, life support or military devices without written permission from RMV ELECTRONICS INC.*
*The material enclosed in this package may not be copied, reproduced or imitated in any way, shape or form without the written consent of RMV ELECTRONICS INC. This limitation also applies to the firmware that the Integrated Circuits in this package might contain.*

***WARRANTY:*** *RMV ELECTRONICS INC. will replace, free of charge, faulty components in this package with the exception of the Integrated Circuits it might contain, for a period of 6 months after the date of purchase.*

## Chapter 1 INTRODUCTION

The ST Stepper Motor Control Board Series is designed for multi stepper motor control through a PC or a terminal. The boards come with drivers for 2 (ST2), 3 (ST3) or 4 (ST4) simultaneous steppers. Acceleration and deceleration modes are supported. Hardware stop pins are available for left and right limits or Home buttons. The boards also provide analog and digital inputs, interrupt inputs and a PWM signal output. Interface to a PC can be done via RS232 or via RS485 used in conjunction with the RMV I/O-485 board. The latter can be configured in an up to 32 board network and therefore up to 128 steppers can be sequentially or simultaneously controlled from one PC serial port. An RMV RS232/RS485 converter makes the network transparent to an ordinary PC serial port.

Boards come with a DLL to allow easy interfacing to Windows® programming languages and software example.

This manual assumes that you have already read the ITC232-A manual. It is also recommended that you read the IO-485 manual should you use the ST boards in a network.

## Features

1. Control of up to 4 stepper motors independently.
2. Motor driving power up to 2 Amp, 50V.
3. Acceleration and deceleration modes are supported.
4. Communication with PCs through RS-232 from 300 to 115200 Bauds.
5. Multiple boards in a network may be controlled through RMV I/O-485 board
6. 8 TTL compatible I/O lines.
7. 10 channel AD Converter with 8, 10 or 12 bits of resolution.
8. One 10 - 10000 Hz, 0 - 100 % duty cycle Pulse Width Modulation channel.
9. High and Low interrupt lines.
10. Driver software (16 and 32 bit DLL) for Visual C (Microsoft or Borland), Delphi and Visual Basic is available.

## Quick Start

The following is a brief description of how to set up the board. Please refer to the connections diagram of the stepper control board in the Appendix for the connection sites.

Step 1. Connect the provided serial cable with 9 pin male and 25 pin female connectors, to the DB9 connector on the board and the serial port of your PC.
Step 2. Plug a board power supply (6.5 to 12V, 500mA recommended) into the power jack (J1) on the board. The red LED will light if there power to the board.
Step 3. Insert a second power transformer into the motor power lines of the PM1 screw terminal block. The rating of this transformer depends on the exact specifications for the stepper motors to be driven. Please refer to your stepper motor manual.
Step 4. If more than one stepper motor is to be controlled, jumper wires must be connected to run power to each driver which requires power.

a. For motors which are less than 1A: If two stepper motors are to be controlled, a jumper wire is run from the positive terminal of PM1 to the positive terminal of PM2. If a third stepper motor is required, an additional jumper wire is run from the positive terminal of PM2 to the positive terminal of PM3. To control a fourth stepper motor a third jumper wire is run from the positive terminal of PM3 to the positive terminal of PM4. Jumper wires do not have to be run between the negative terminals of the PMx terminals which are driving a stepper motor as they share a common ground on the board. The negative terminal only needs to be connected between the motor power supply and one of the PMx terminals being used.

b. For stepper motors over 1A: If your system uses steppers motors greater than 1 A, it is recommended you connect the power to the motors in the following configuration: For each stepper motor you wish to control, a wire is run from the positive terminal of each PMx terminal which is driving a stepper motor to a common point. Twist the free ends of these wires together and connect this to the positive terminal of the power supply. The same must be done for the negative terminals of each PMx terminal which is driving a stepper motor. Naturally, the common point of the wires leading from the negative terminals is attached to the negative terminal of the power supply.

c. For stepper motors of differing power requirements: It is recommended that you connect the positive terminal of each PMx terminal which is driving stepper motor(s) with the same power requirements (i.e. same voltage) to the positive terminal of the suitable power supply as described in 'b' above. The negative terminals are also connected to the negative terminal of the same power supply.

The board is now fully connected, powered and ready to run.

Insert the provided Stepper Motor Controller Board diskette and transfer the self-extracting zip file ('Install.exe') onto your hard drive. Double click on the Install file to unzip the file. Double click on the 'Demo.exe' file to run the demonstration program. The default settings of Com Port 2 and 9600 baud should work in most situations.

For a demonstration of the independent control of stepper motors, go to the 'Control' menu and select 'Motor Control Panel'. Information on the different control functions may be found in the 'Help' menu.

# Chapter 2 FUNCTION DESCRIPTION

## Stepper Motor Control

### Stepper Motor Control IC

The heart of the stepper motor control is the CY500 IC. This IC is a standard 5-Volt, 40 pin LSI device configured to control any 4-phase stepper motor. It will interface to any computer using asynchronous parallel TTL input and provides numerous TTL inputs and outputs for auxiliary control and interface.

The DB0~6 pins are connected to Port B of the ITC232-A chip for accepting commands. The DB7 pin is ground so that the CY500 works in ASCII mode. The CONTROL pin enables the L298 driver. The ABORT, TRIGGER and RESET pins are controlled by the ITC232-A chip, which starts and stops the motors.

The ST4 board consists of four CY500 ICs (U8, U9, U10 and U11). They are operated independently. Thus, the ST board can control a total of 4 stepper motors. To address the four CY500 ICs, two pins (PC0, PC1) of Port C of the ITC232-A are used, and are decoded by a 74LS139. The RESET, WR, and ABORT control pulse are latched in each 74116 when it is selected by a 74LS139. The two address pins (PC0, PC1) are also connected to the selection pins of the 74LS153, which read back the BUSY/RDY pins of the CY500 ICs and read back the HOME BUTTON status.

The TRIGGER pins of CY500 ICs are connected to the 74LS373 which latches the data from PB0~PB3 on the ITC232-A chip. When the TRIGGER pin is high, the motor will not turn, and when it is low the motor will turn.

The CY500 ICs are reset at power up, and can also be reset individually by a reset signal through PC2 of the ITC232-A.

To stop a motor or to set the home position of a motor, a six-pin header strip and a circuit are used for each CY500 IC. The function is achieved by controlling the ABORT, and EXT START/STOP pins. The abort signal comes from two sources. One is from PC4 of the ITC232-A. The other is from the header strip (JH1, JH2, JH3, or JH4) by connecting a push button or external logic signals.

### The Driver IC – L298

All the stepper motors are driven by an L298 which is a high voltage, high current, dual full-bridge driver. It can handle up to a 2 Amp current. The power supply for motors can be up to 50V. For more powerful stepper motors, a simple H bridge driver circuit can be added as shown in the figures at the end of this manual.

## Communication with PCs

The ST can be controlled directly via a standard serial port or attached to our I/O-485 stations. The latter configuration allows the user to control up to 32 ST boards from the same serial port, with synchronic triggering of all motors if required.

### ITC232-A

An ITC232-A is used as the interface between the PC and the CY500 chips. Only Port B and Port C, and the serial port are used for this purpose. The communication between the ST board and the PC is implemented by connecting the RS232 port of PCs and the serial port of the board driven by the MAX232 IC. The baud rates supported are from 300 to 115200.

PB0.6 on the ITC232-A are used for sending commands to CY500 in ASCII mode when PC7 is low. PB0.3 are also used to trigger the CY500 when PC7 is high. PB7 is used for reading the Home Flag which will be set to 1 when the moving device touches the Home Button.

PC0 and PC1 are used to address the four CY500 ICs. The combinations of PC0 and PC1 decode the four CY500 IC's addresses. PC2 resets the CY500 IC addressed by PC0 and PC1. PC3 sends a WR signal to the CY500 addressed by PC0 and PC1. PC4 is used for sending a stop signal to the CY500 addressed by PC0 and PC1. The status of the BUSY/RDY pins of the CY500 addressed by PC0 and PC1 is read back through PC5. PC6 resets the A/D converter. When PC7 is low, the 74LS139 is enabled, and the address set by PC0 and PC1 is asserted, while the output of 74LS373 are not affected by its input. When PC7 is high, the 74LS139 are disabled, and the address is not asserted, while the trigger signals from PB0~PB3 come through the 74LS373 to each CY500.

### Networking with an I/O-485 Board

The ST board can attach to the I/O-485 board so that it can be controlled with an RS485 network. To do so, remove the ITC232-A chip on the ST board, and connect the 40 pin header strip (JP6) on the ST to the 40 pin header strip (CON1) on the I/O-485 board. The ST board is controlled by the U1 (ITC232-A chip) on the I/O-485 board. To operate the I/O-485 board, please refer to the I/O-485 board manual.

## Other Features

### A/D Converter

The ST board supports 10 analog to digital conversion channels. The resolution of the conversion depends on the ADC used (MC145041, 8 bits; MC145051, 10 bits; TLC2543, 12 bits.) They work in a similar manner. The SPI port on the ITC232-A is used to control the ADC. For more detail on using the ADC, please refer to the ITC232-A manual.

### IRQ, PWM and I/O Ports

The ST board supports two Interrupt inputs (IRQL and IRQH), a Pulse Width Modulation (PWM) output, and 8 I/O TTL compatible pins (Port A of the ITC232-A). All these resources belong to the ITC232-A.

## Software

A dynamic link library (DLL) and a demonstration program are provided for the ST board. The library provides all the necessary subroutines for controlling this board. The program can be run under Windows 3.1™, Windows 95™ or Windows NT™, and can be called from Delphi, Visual Basic and C++.

The demonstration program provides examples: on controlling the board with the library; demonstrating the effects of all subroutines; and also provides a control panel to facilitate board control.

# Chapter 3 OPERATION OF THE ST BOARD

## Connections

### Connections of Stepper Motors

There are four connectors (JM1, JM3, JM5, and JM7) on the ST4 board, i.e. four 4-phase stepper motors can be attached to and controlled by the board. The stepper motors connected to JM1 are controlled by the first CY500 IC (U8); the one connected to JM3 is controlled by U9; the one connected to JM5 is controlled by U10; the one connected to JM7 is controlled by U11.

### Power Supply for the Board

A 7805 (+5 V) regulator provides the power to the board. A 6.5-9V, 500mA adapter is recommended. The power supply should be connected to the Stereo-jack (J1) or the connector (PWR1) on the ST board.

### Power Supply for the Stepper Motors

The power supplies for the motors depend on the stepper motors used. Connector PM1 is for the stepper motor connected to connector JM1. PM2 is for JM3. PM3 is for JM5. PM4 is for JM7. All the power supplies have the same ground.

## Communication with PCs

The DB-9 connector (P1) on the ST board should be linked to the serial port of a computer or terminal. Communication settings for the ITC232-A and for the computer should be the same. These settings include Baud rate, data bits, stop bits, parity, and flow control. The power up configuration of the ST board is 9600-Baud rate, 8 data bits, 1 stop bit, no parity and no flow control. The Baud rate can be changed via software to any standard value between 300 and 115200 Baud.

The jumper JP1 and jumper JP2 allow swapping of the TxD and RxD lines to accommodate both straight and "null modem" cables.

## Controlling Stepper Motors

Read this chapter carefully if you are planning on developing code in DOS or for the MAC to access the ST directly. If you are working in Windows, use the DLL provided (see Chapter 4).

### Initializing the Board

a. Configuring ITC232-A ports, unlock home buttons

Port B of the ITC232-A should be configured as
pcb127 (0111 1111),
i.e. the pins of PB0..6 are configured as outputs, and BP7 is configured as an input. Port C should be configured as
pcc223 (1101 1111),
i.e. PC0..4 and PC6..7 are configured as outputs and PC5 is configured as an input.

This is the most important part of the initialization process. If the ITC232-A ports are not properly configured, the ST board would not work.

b. Reset the CY500 ICs, unlock home buttons

When the ST board is powered up, all the chips on the board will be reset, all the BUSY/READY pins of the CY500 ICs will be high, (i.e. in READY status), and the home button status read from the ZB pin (the second output pin) of the 74LS153 will be high which means the motor is not in the home position. To make sure the configuration is correct, we may need to configure this by sending the following commands through ITC232-A:

```
pwb15 (0000 1111)
pwc156 (1001 1100) ;        this will set all the TRIGGER pins to high, so that the
                            motors will not move.
pwc8   (0000 1000)
pwc9   (0000 1001)
pwc10 (0000 1010)
pwc11 (0000 1011) ;         these will lower all the RESET and ABORT pins.

[Delay about 100ms];        this requires software be written by the user to
                            perform this delay.
pwc24 (0001 1100)
pwc25 (0001 1101)
pwc26 (0001 1110)
pwc27 (0001 1111)
```

As explained in section 2.1.1, the two least significant bits of the Port C of the ITC232-A are used for addressing the four CY500 ICs.

When one of the home buttons is set at home position, the ABORT pin of the corresponding CY500 will be kept low until the PC4 of the ITC232-A is low and the address of the CY500 is set.

### Sending Commands to the CY500 ICs

Because the CY500 ICs are working in ASCII mode, all the CY500 commands sent through Port B of the ITC232-A should be in ASCII string. For example, to set 100 steps, the command string is "N 10<return>" containing 5 bytes data, where <return> means the ASCII code '13'. To accept one data byte, a low strobe from the WR pin of the CY500 is needed. Thus, to send this string to one of the four CY500 ICs, the following code should be sent out to the ITC232-A (It is assumed in the example that the string is sent to the second CY500 (U9) ).

```
pwcb00011101   ;        set the address
prb            ;        read Port B
if (PB7 is high) then (continue);   When PB7 is high, the CY500 is ready.
Pwb78          ;        '78' is the ASCII code of the letter N.
pwcb00010101   ;        set the PC3 to low, i.e. set the WR pin low.
prb            ;        read Port B
if (PB7 is low)  then (continue) else (wait);
pwcb00011101   ;        set the WR pin high.
prb            ;
if (PB7 is high)  then (continue) else (wait);
pwb20          ;        '20' is the ASCII code of the space.
pwcb00010101   ;        set the PC3 to low, i.e. set the WR pin low.
prb            ;        read Port B
if (PB7 is low)  then (continue) else (wait);
pwcb00011101   ;        set the WR pin high.
prb            ;
if (PB7 is high)  then (continue) else (wait);
```

```
pwb31              ;        '31' is the ASCII code of  1.
pwcb00010101       ;        set the PC3 to low, i.e. set the WR pin low.
prb                ;        read Port B
if (PB7 is low)    then (continue) else (wait);
pwcb00011101       ;        set the WR pin high.
prb                ;
if (PB7 is high)   then (continue) else (wait);
pwb30              ;        '30' is the ASCII code of  0.
pwcb00010101       ;        set the PC3 to low, i.e. set the WR pin low.
prb                ;        read Port B
if (PB7 is low)    then (continue) else (wait);
pwcb00011101       ;        set the WR pin high.
prb                ;
if (PB7 is high)   then (continue) else (wait);
pwb13              ;        '13' is the ASCII code of  carriage return.
pwcb00010101       ;        set the PC3 to low, i.e. set the WR pin low.
prb                ;        read Port B
if (PB7 is low)    then (continue) else (wait);
pwcb00011101       ;        set the WR pin high.
```

### Moving the Stepper Motors

To move the motor, several steps should be followed.

a. enable the L298 (or power up the L298) by sending command "B" to the CY500.

b. set the TRIGGER pin of the CY500 to low, so that the CY500 is triggered. The code for doing this, is as follow:

```
pwb0
pwcb10011100
pwcb00011100
```

c. send CY500 commands like "N xx", "R xx", "F xx", "+", "S xx" to configure the speed, Number of steps, direction, acceleration, where "xx" means any number.

d. send CY500 command of "G", to start moving the stepper motor.

The sequence of the above steps is not important. Step b can be the last step, i.e. when the motor is ready to go, set the trigger.

### Stopping the Stepper Motors

There are several ways to stop a moving stepper motor:

a. Auto stop: When the motor finishes the number of steps set by the command "N xx", the motor will stop automatically.

b. Abort: When the ABORT pin of the CY500 goes low, the motor will be stopped. There are two ways to set the ABORT bit low, one is by software by setting PC4 pin of the ITC232-A to low. Another way is by pressing home button. When the home button is set, the ABORT pin will be set low and kept low until the PC4 pin is set low.

c. Un-trigger: When the TRIGGER pin of the CY500 is high, the moving motor will be stopped, however when the pin is low, motion will be resumed.

d. "Q" command: When the CY500 command "J" is used, the "Q" command will stop the motor.

### Triggering the Stepper Motors

As explained before, to send the trigger signal to the CY500s, PC7 should be high, and the 4 least significant pins of Port B are used as triggers. PB0 is for the first CY500 (U8), PB1 is for the second (U9), PB2 is for U10 and PB3 is for U11. When PB0 is low, the CY500 (U8) is

triggered ( (the same for PB1, PB2 and PB3). To start two or more motors at the same time, you can un-trigger the motors first, then configure the motors so that they are ready to go and finally trigger them.

### Home Buttons

Every CY500 IC has a "Home button" circuit which sets the ABORT pin low (abortion) when the motor reaches home position, and a 6 pin header strip (JH1 for U8, JH2 for U9, JH3 for U10 and JH4 for U11) which connects with up to two push buttons or two external stop signals. Examples of the connection for the buttons can be found in the figures at the end.

## A/D Conversion

### Reset

When the board is powered up, the A/D converter will be reset automatically. You can also reset the ADC by setting PC6 to high for a period of 100 ms.

### Initialization, Getting Conversion Results

In the ITC232-A manual, the operation of initializing the ADC and getting conversion results from the ADC is explained in detail. Please refer to the manual for more information regarding this topic.

---

# Chapter 4 PROGRAMMING THE ST VIA THE PROVIDED DLL

The DLL for the ST board is designed to facilitate the work of programming the ST Board from Delphi, Visual Basic, and C++ in Windows.

The functions in this DLL can be divided into three major categories. There are port functions with the prefix 'Port' for serial communication, ITC functions with the prefix 'ITC' for controlling the ITC232-A chip and Motor functions with the prefix 'Motor' for step motor controlling.

## Using the DLL with Different Programming Languages

The DLL for the ST board is written in Delphi, however, it can be used equally well with Visual Basic and C. In the Demo program, the declarations for all the functions required by your program are listed and they can be cut and pasted into your own code. In Appendix A of this manual, all the functions are explained in detail.

## Controlling the Motors

### Establishing the Connection Between a PC and the ST Board

First, the serial port must be configured and opened. To configure the serial port, you call either the '***PortConfigure***' or the '***PortConfigureForm***' functions. The difference between these two functions is that the latter provides a dialog form with the DLL to set the required parameters. '***PortConfigure***' does not open a configuration form; instead, the COM port number and the Baud rate are passed as function parameters. These two functions are optional before you open the

serial port by calling '***PortOpen***' because '***PortOpen***' will use default parameters if the configuration function is not called. The default parameters are COM2, 9600 Baud Rate, 8 Data bits, 1 Stop bit, no Parity bit. After the serial port is opened, you can always call the configuration functions to change the Baud Rate.

### Initializing the ST Board

After the serial communication is established, you can send commands to control the motors and other features provided in the board by calling Motor functions and ITC functions. The first function that should be called is '***MotorInit***', which will configure the I/O Ports on the ITC232-A, reset each CY500, set all the motors in pause status and unlock all home buttons.

### Configuring the Motors

Every CY500 IC should be configured before making a motor turn. The configuration includes setting speed, direction, and number of steps. These can be done by calling the '***MotorSpeed***', the '***MotorDirection***' and the '***MotorNumberStep***' functions. The first parameter in these functions specifies which CY500 IC you are talking to.

### Turning and Stopping the Motors

Either '***MotorGo***' or '***MotorJog***' will make a specified motor turn. The difference is that '***MotorGo***' will make the motor turn the number of steps specified by '***MotorNumberStep***'. Instead, '***MotorJog***' will make the motor step until you abort its stepping by calling '***MotorStop***' or the mechanism comes in contact with the home button.

There are several ways to stop motor stepping. '***MotorPause***' will make a stepping motor pause, and '***MotorContinue***' will resume the stepping. These two functions are a subset of the function '***MotorTrigger***' since they set one of the four parameters of this function. When one of the parameters is zero, the corresponding motor will pause (untriggered). Setting this parameter to one awakens the motor to continue stepping (triggered). This is handy for starting two or more motors simultaneously. When the board is initialized by '***MotorInit***', all the motors are in pause mode. Either '***MotorContinue***' or '***MotorTrigger***' should be called to stop the pause.

'***MotorStop***' will abort motor stepping, leaving the motor position step counter undefined. The position counter records the position of a motor when the motor finishes a action. Aborting an action or using '***MotorJog***' will destroy this information and set it to an undefined state.

'***MotorReset***' or '***MotorInit***' will also stop a motor by resetting the motor control chip. These are not recommended for this purpose. Both functions will set the position count to zero.

### Motor Homing

Usually, the homing technique is needed to know the physical position of a motor. On the board, there are four 6-pin connectors for you to attach your Home button for each motor. When a motor touches its 'Home' button, it stops immediately. Its position count is set to zero. And it is locked in the position until you call the '***MotorGetHomeFlag***' function to get the home flag and release the lock.

### Motor Command

The '***MotorCommand***' function provides a way that you can send all motor commands supported by the CY500 to the IC. Similarly, the '***ItcCommand***' function lets you to send commands supported by the ITC232-A. Using them, you can build your own stepper motor control functions.

### Error Message

Most functions in the DLL will return 1 when the process of the function is successful and 0 when the process of the function fails. In the latter case, you can call the '***ItcGetErrorMessage***' (string) function or call the '***ItcGetError***' (error number) function to get the error code.

## A/D Conversion

Performing AD conversions using the DLL is quite simple. Before any conversion, you may need to call the '***ItcResetADC***' function to reset the ADC. Then call the '***ItcAnalog8***' or '***ItcAnalog10***' or '***ItcAnalog12***' function for 8 bits, 10 bits and 12 bits respectively with the channel number as a parameter. Finally call the '***ItcGetAnalogValue***' to get the result.

## Network Functions

The DLL provides two functions to manipulate the I/O-485 network card. They are the '***ItcInitialize485***' function which initializing the I/O-485 card, and the '***ItcAddress485***' function which selects a station and an ITC232-A chip in the station.

# Appendix A

## DLL Functions

The Stepping Motor Control DLL is written in Delphi, for Delphi users, all the functions are available except two, ***ItcGetErrorMessagevb*** and ***MotorGetErrorMessagevb***, which return strings in Visual Basic format, consequently they are for Visual Basic only. Alternatively, Delphi users and C users can use ***ItcGetErrorMessage*** and ***MotorGetErrorMessage***.

---

function ***PortOpen*** (cPort:Integer):Integer;

cPort: Specifies the COM port.

Open the serial communication port.

**Returns:**

Returns 1 if successful. Otherwise returns 0.

---

function ***PortClose***:Integer;

Close the serial communication port.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ItcGetError or ItcGetErrorMessage.

---

function ***PortConfigure***(*cPort*:Integer; *cBaud*:LongInt):Integer;

The ***PortConfigure*** function Configures the serial communication port.

**Parameters:**

| | |
|---|---|
| *cPort*: | specifies the COM port.<br>*cPort*=0 for COM0, 1 for COM1 and so on. |
| *cBaud*: | specifies the baud rate.<br>*cBaud*=9600 for 9600 Baud , etc. |

**Returns :**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***ItcGetPort***:Integer;

This function returns the current serial communication port number.

**Returns:**

Returns 0 for COM0, Returns 1 for COM1 and so on.

---

function ***ItcGetBaudRate***:LongInt;

This function returns the current Baud Rates.

**Returns:**

Returns 9600 for 9600 baud rates, return 19200 for 19200 baud rates and so on.

---

function ***ItcChangeBaudRate***(*NewSpeed*:Longint):Integer;

This function changes the baud rate on both the PC serial port and the ITC-232A.

**Parameters:**

*NewSpeed*: specifies the new baud rate required.
e.g. *NewSpeed* = 9600.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***ItcCommand*** (*Order*:string):Integer; (for Delphi and C users)

The ItcCommand function sends the ITC-232A command set to ITC-232A chip.

**Parameters:**

*Order:* specifies the command being sent to the ITC-232A chip.
e.g. 'pra'.

**Returns :**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***. The message returned from ITC-232A can be obtained by calling ***ItcGetValue***.

**Comments:**

For Visual Basic and C/C++ user, use the alternative function ***ItcCommandvb***.

---

function ***ItcCommandvb***(*Order*:PChar):Integer; (for Visual Basic users)

This function sends ITC-232A command set to ITC-232A chip.

**Parameters:**

*Order:* specifies the command being sent to the ITC-232A chip.
e.g. 'pra'.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage*** (or ***ItcGetErrorMessagevb*** for VB user). The message returned from ITC-232A can be obtained by calling ***ItcGetValue***.

---

function ***ItcGetValue***: LongInt;

The ***ItcGetValue*** function returns the value sent back from the ITC232-A (if any).

---

function ***ItcGetError***: Integer;

This function returns the error code when it is fail to call the functions. The Error Codes are as follows:

1 = ITC232-A reports error: Syntax error
2 = ITC232-A reports error: Port must be configured or enabled first
3 = ITC232-A reports error: Command not allowed in current configuration
4 = ITC232-A reports error: No such port
5 = ITC232-A reports error: Value out of range or syntax error
6 = ITC232-A reports error: Pin configured as an output
7 = ITC232-A reports error: Time out error
8 = ITC232-A reports error: Frequency too high for required duty cycle
9 = ITC232-A reports error: Baud rate not supported
10 = ITC232-A reports error: Port D is always a 4 bit input port
11 = ITC232-A reports error: SPI requires pin PD3/PS_VDD always high, change and try again
20 = COM port not supported
21 = COM port not found
22 = Baud rate not supported
23 = Attempt to close a COM port which is not open
24 = The COM port returned more than 255 characters
25 = Commands cannot be sent before a COM port is open
26 = COM port timed out
27 = COM port already open
28 = not used
29 = Returned value out of range
30 = I/O-485 address out of range (0..31)
31 = I/O-485 IC out of range (0, 1 or 3 {or BROAD})
32 = I/O-485 Broadcast parameter must be 1 or 0
40 = Motor Control Chip is NOT Ready
41 = Can not Write to Motor Control Chip
42 = Speed out of range
43 = Motor Abort Fail

---

function ***ItcGetErrorMessage***:pchar;

This function returns the error message when it fails to call the functions. The alternative function for the Visual Basic user is ***ItcGetErrorMessagevb***.

---

function ***ItcIntStatusH***:Integer;

This function returns the status of the IRQH interrupt from the ITC-232A, and also resets the status.

**Returns:**

Returns 1 if the IRQH interrupt is assert. Otherwise 0.

---

function ***ItcIntStatusL***:Integer;

This function returns the status of the IRQL interrupt from ITC-232A, and also reset the status.

**Returns:**

Returns 1 if the IRQL interrupt is assert. Otherwise 0.

---

function ***ItcAnalog8***(*Channel*:Integer):Integer;

This function forces the 8-bit ADC (MC145041) to convert an analog value. This value can be returned by the function ***ItcGetAnalogValue***.

**Parameters:**

*Channel* : *Channel*=0 for Channel 0; *Channel*=1 for Channel 1 and so on.
There are 10 Channels supported by the board.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***ItcAnalog10***(*Channel*:Integer):Integer;

This function forces the 10-bit ADC (MC145051 or TLC1543) to convert an analog value. This value can be returned by the function ***ItcGetAnalogValue***.

**Parameters**

*Channel* : *Channel*=0 for Channel 0; *Channel*=1 for Channel 1 and so on.
The ST supports 10 Channels.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***ItcAnalog12***(*Channel*:Integer):Integer;

This function forces the 12-bit ADC (TLC2543) to convert an analog value. This value can be returned by the function ***ItcGetAnalogValue***.

**Parameters:**

*Channel* : *Channel*=0 for Channel 0; *Channel*=1 for Channel 1 and so on.
The ST supports 10 Channels .

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***ItcResetADC***:integer;

The function resets the ADC in the board.

**Returns:**

Returns 1 if successful. Otherwise returns 0. The Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***ItcGetAnalogValue***:LongInt;

The function returns the ADC converted value.

---

function ***ItcInitialize485***: Integer;

Initialize the I/O-485 board and set to station 0 and chip U1.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***ItcAddress485***(*Station*,*IC*,*Brcst*:Integer):Integer;

This function is for selecting an ITC-232-A chip specified by parameter 'IC', in a I/O-485 board specified by parameter 'Station'.

**Parameters:**

*Station*: specifies station number from 0 to 31.
*IC*: *IC*=1 for U1, *IC*=2 for U2
*Brcst*: specifies broadcast mode.
*Brcst*=0 for non-broadcast mode.
*Brcst*=1 for broadcast mode.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

**Comments:**

For more details about the I/O-485, please refer to its manual.

---

function ***MotorInit***:Integer;

This function initializes the ST board. It configures the ITC-232A chip on the ST Board, resets the Position Counter and resets all the motor controller ICs.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function *MotorInit*:Integer;

This function initializes the ST4 board. It configures the ITC-232A chip on the ST4 Board, resets the Position Counter, and resets all the motor controller ICs.

Returns:

Returns 1, if successful. Otherwise returns 0. Error Message can be obtained by calling ItcGetError or ItcGetErrorMessage.

---

function ***MotorNumberStep***(*WhichMotor*:Integer;*Number*:LongInt):Integer;

This function sets the Number of steps for the motor to turn.

**Parameters:**

*WhichMotor*: specifies the motor to be configured.
0: Motor 1
1: Motor 2
2: Motor 3
3: Motor 4
*Number*: specifies the Number of Step range from 1 to 65536.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***MotorSpeed***(*WhichMotor*:Integer;*Speed*:double;*AccStep*:LongInt;*CrysFreq*:double):Integer;

This function configures the speed and the acceleration of the motor.

**Parameters:**

*WhichMotor*: the same as the one in the function MotorNumberStep.
*Speed*: specifies the motor speed in Steps/Second.
*AccStep*: specifies the number of steps within which the motor finishes acceleration.
e.g. *AccStep*=50 means the Motor reaches the full speed specified by the parameter '*Speed*' after 50 steps.
If *AccStep*=1 or 0, then the motor is in non-acceleration mode.
*CrysFreq*: specifies the frequency of the crystal for CY-500. It should be 6.0 for the 6MHz crystal provided with the board.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

**Comments:**

When *AccStep*>1, the motor is in acceleration mode. To set back to non-acceleration mode, set *AccStep* to 1 or 0.

---

function ***MotorDirection***(*WhichMotor*:integer;*Cw*:integer):Integer;

This function sets the turning direction of the motor specified by ***WhichMotor*** parameter.

**Parameters:**

*WhichMotor*: the same as the one in the ***MotorNumberStep*** function.
*Cw*: specifies the direction (depending on the wiring of the phases ).
0: Counterclockwise
1: Clockwise.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***MotorGo***(*WhichMotor*:Integer):Integer;

This function makes the motor turn N steps. N is specified by the ***MotorNumberStep*** function.

**Parameters:**

*WhichMotor*: same as for ***MotorNumberStep***.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

**Comments:**

The motor will not turn if the trigger mask is zero, or the MotorPause function is called. In this case, call the ***MotorContinue*** function or set the trigger mask to one by calling the ***MotorTrigger*** function, to turn the motor.

---

function ***MotorJog***(*WhichMotor*:Integer):Integer;

This function turns the motor an infinite number of steps.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage.***

---

function ***MotorStop***(*WhichMotor*:Integer):Integer;

This function stops the motor, and sets the position counter to 2147483647.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***..

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***MotorPause***(*WhichMotor*:Integer):Integer;

This function stops the motor temporarily. It is equivalent to setting the trigger mask to zero.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage.***

---

function ***MotorContinue***(*WhichMotor*:Integer):Integer;

This function is opposite to the ***MotorPause*** function.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***MotorPowerOn***(*WhichMotor*:Integer):Integer;

This function turns on the motor power supply.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.

**Returns**:

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***MotorPowerOff***(*WhichMotor*:Integer):Integer;

This function turns off the motor power supply. When the motor is turning, the power cannot be turned off by this function.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

---

function ***MotorGetHomeFlag***(*WhichMotor*:Integer):Integer;

This function returns the Home Flag, sets the Home Flag to zero, releases the motor which is locked in the Home Position, and sets the position counter to zero if the motor touched its Home button.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.

**Returns:**

Returns 1 if the motor touched its Home Button. Otherwise returns 0.

---

function ***MotorGetTrigMask***: Integer;

This function returns the Trigger mask for the four motors.

**Returns:**

| | Motor 4 | Motor 3 | Motor 2 | Motor 1 |
|---|---|---|---|---|
| 0: | N-T | N-T | N-T | N-T |
| 1: | N-T | N-T | N-T | T |

| | | | | |
|---|---|---|---|---|
| 2: | N-T | N-T | T | N-T |
| 3: | N-T | N-T | T | T |
| 4: | N-T | T | N-T | N-T |
| 5: | N-T | T | N-T | T |
| 6: | N-T | T | T | N-T |
| 7: | N-T | T | T | T |
| 8: | T | N-T | N-T | N-T |
| 9: | T | N-T | N-T | T |
| 10: | T | N-T | T | N-T |
| 11: | T | N-T | T | T |
| 12: | T | T | N-T | N-T |
| 13: | T | T | N-T | T |
| 14: | T | T | T | N-T |
| 15: | T | T | T | T |

Where 'N-T' means the motor is not triggered and 'T' means the motor is triggered. When the motor is not triggered, the motor will not turn while the MotorGo function is called. This is equivalent to the MotorPause function. When the motor is triggered, the motor will turn if the MotorGo function is called. This is equivalent to the MotorContinue function.

---

function ***MotorGetReadyFlag***(*WhichMotor*:Integer):Integer;

This function returns the motor ready flag. When the motor is not ready, it will not accept any command. When the motor is turning, the motor is not ready.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.

**Returns:**

Returns 1 if the motor is ready to accept command. Otherwise returns 0.

---

function ***MotorGetPosition***(*WhichMotor*:integer):LongInt;

This function returns the value of the motor position counter.

**Parameters*:***

*WhichMotor*: same as in ***MotorNumberStep***.

---

function ***MotorCommand***(*WhichMotor*: Integer; *Cmd*:String):Integer;

This function sends the motor commands to the motor.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.
*Cmd*: the motor command.

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

**Comments:**

Almost all the functions with the prefix 'Motor' are derived from this function. Using this function, you can develop your own motor control.

---

function ***MotorTrigger***(*Motor1*, *Motor2*, *Motor3*, *Motor4*:Integer):Integer;

This function sets the trigger mask.

**Parameters:**

| | |
|---|---|
| *Motor1*: | 0: Motor Not Triggered |
| | 1: Motor Triggered |
| *Motor2*: | 0: Motor Not Triggered |
| | 1: Motor Triggered |
| *Motor3*: | 0: Motor Not Triggered |
| | 1: Motor Triggered |
| *Motor4*: | 0: Motor Not Triggered |
| | 1: Motor Triggered |

**Returns:**

Returns 1 if successful. Otherwise returns 0. Error Message can be obtained by calling ***ItcGetError*** or ***ItcGetErrorMessage***.

**Comments:**

The ***MotorPause*** and ***MotorContinue*** functions are subsets of this function. If you want two or more motors to start simultaneously, you can set them not triggered first, then call the ***MotorGo*** function and set them triggered.

---

function ***MotorReset***(*WhichMotor*:Integer):Integer;

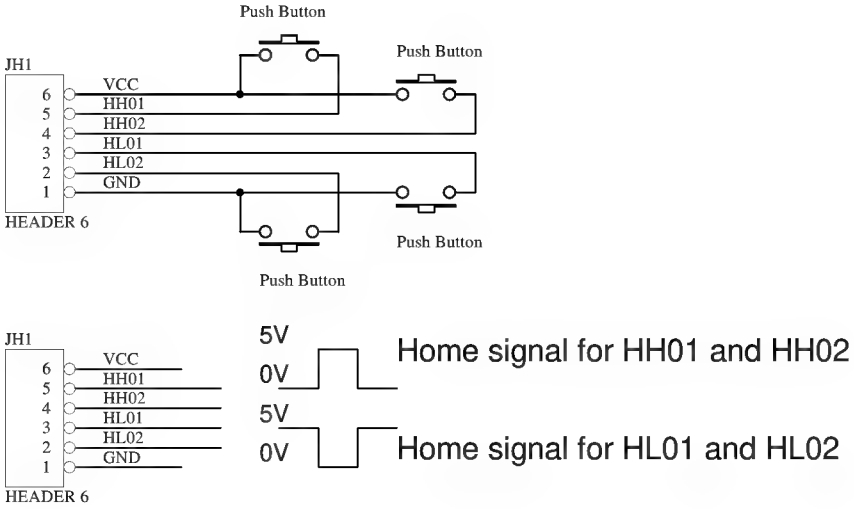
This function performs a hardware reset to the Motor specified by the *WhichMotor* parameter, and sets position counter to zero.

**Parameters:**

*WhichMotor*: same as in ***MotorNumberStep***.

**Returns**:

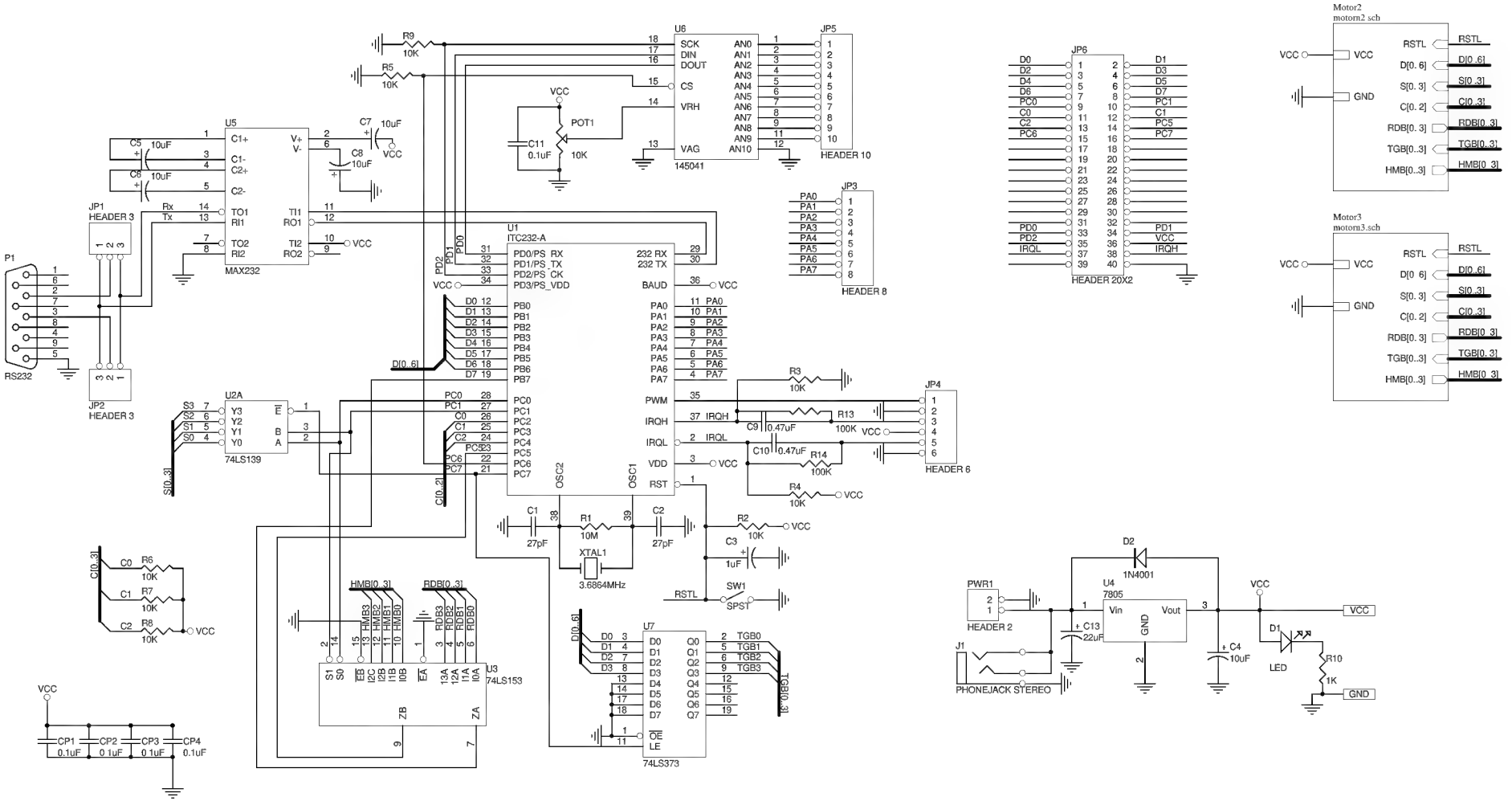
Returns 1 if the motor is ready to accept the command. Otherwise returns 0.




Example for Home Button Connection to JH1(the same for JH2,JH3 and JH4).

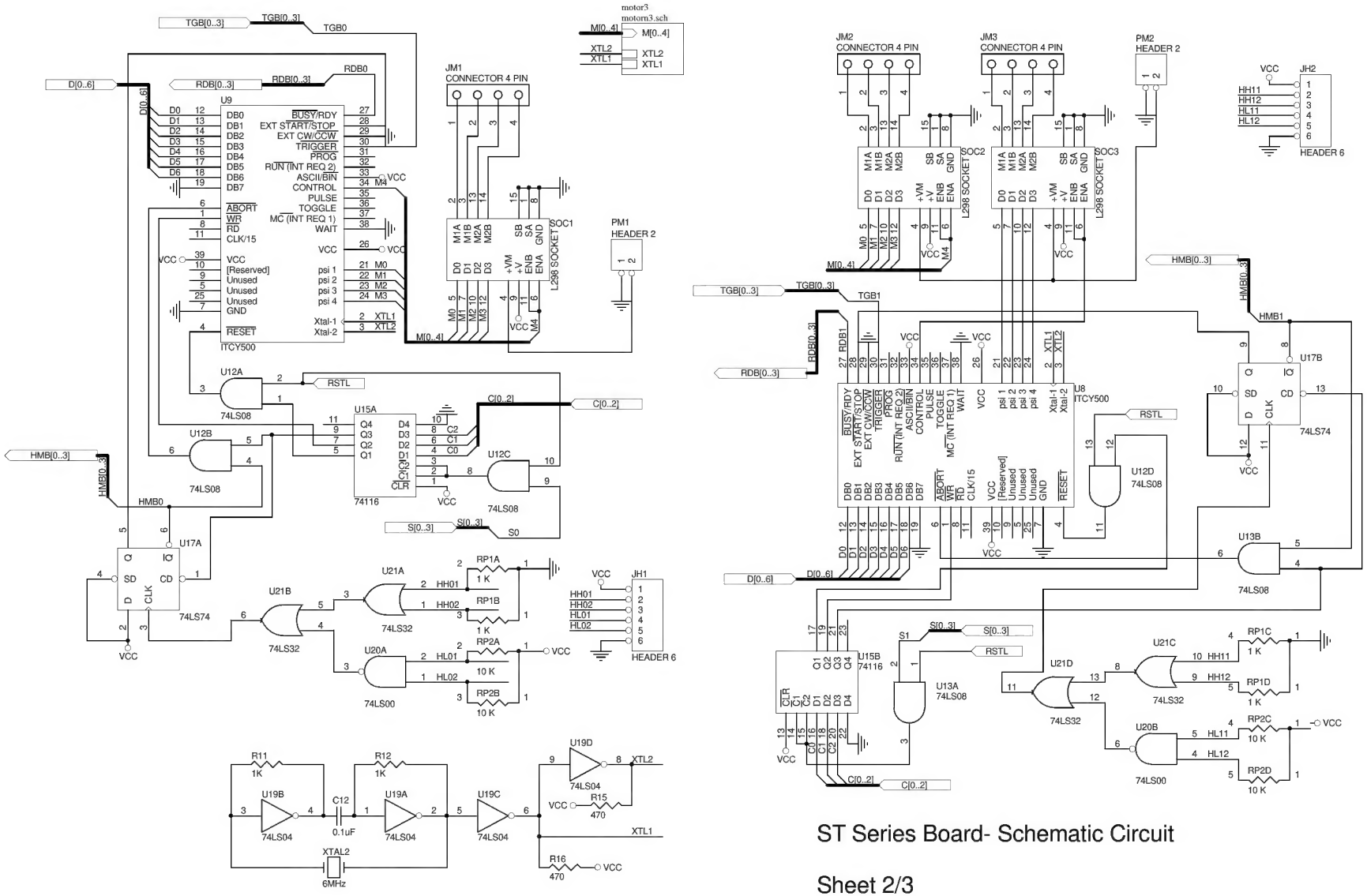
ST Series Board

Limit Switches Connection



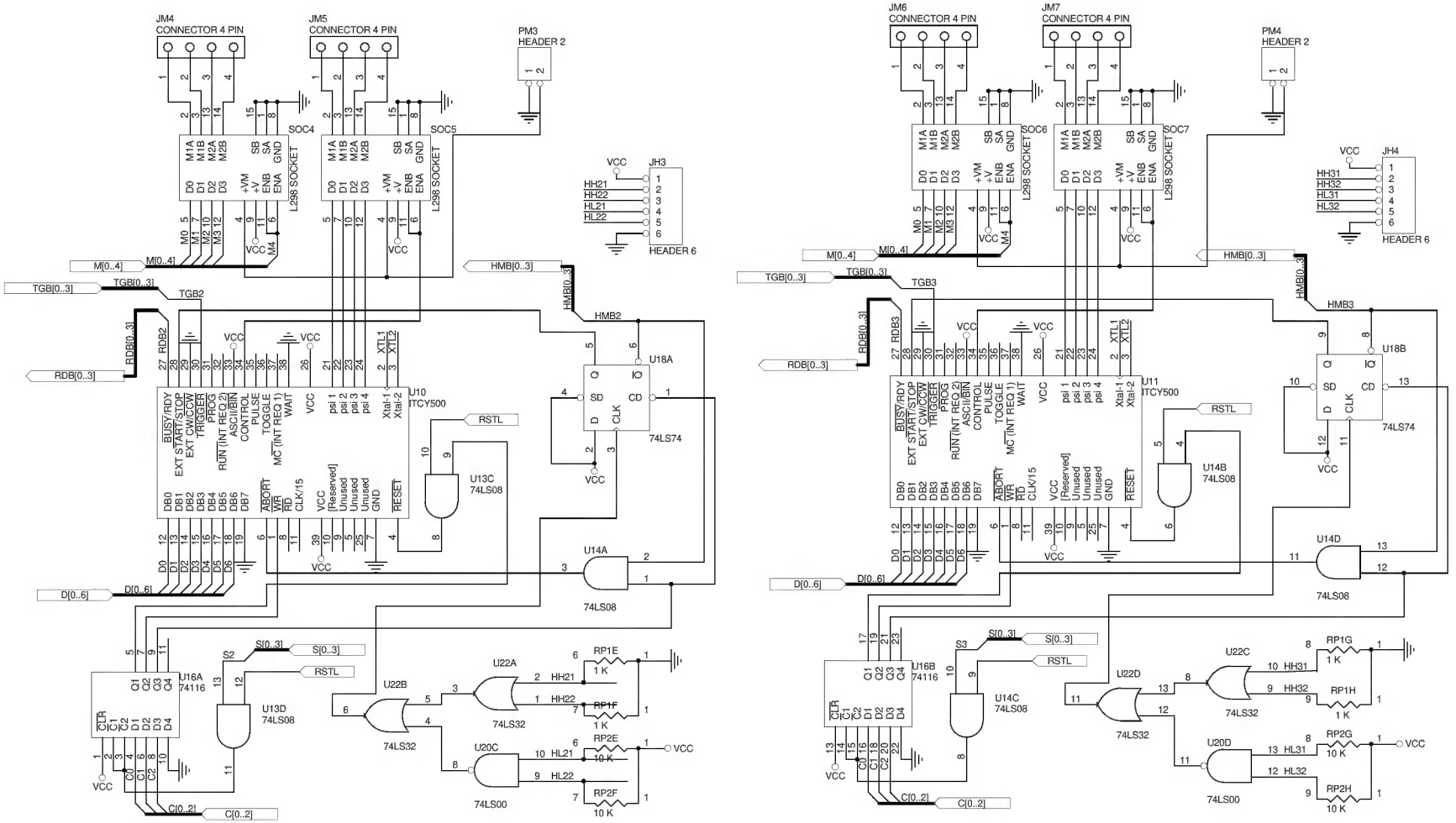

ST Series Board- Schematic Circuit

Sheet 1/3




ST Series Board- Schematic Circuit

Sheet 2/3

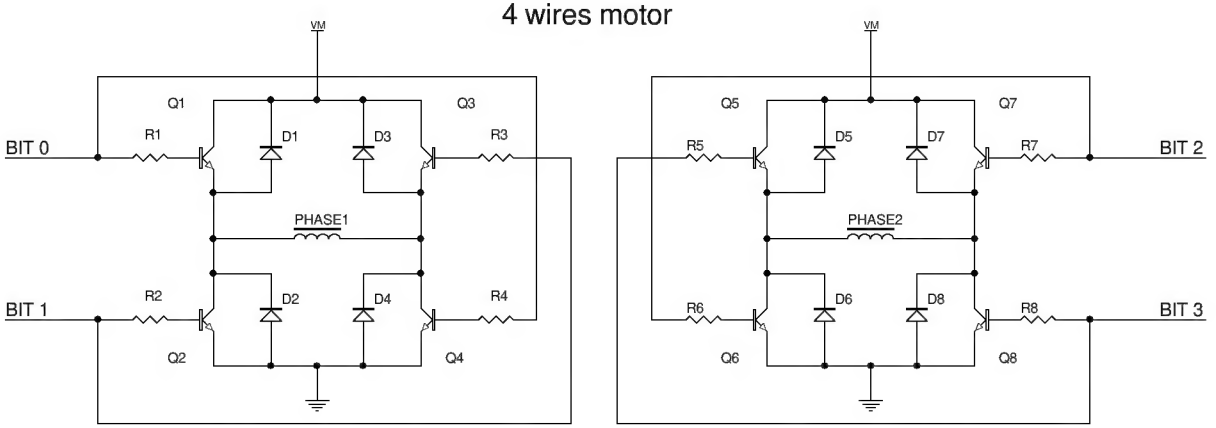



ST Series Board- Schematic Circuit

Sheet 3/3

# BIPOLAR & UNIPOLAR DRIVER CIRCUITS

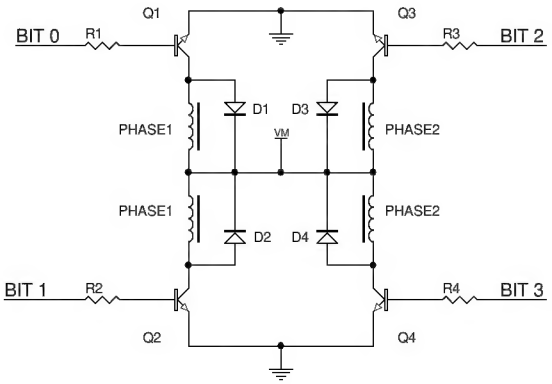
## 4 wires motor




Reffer to
ST-Series Board
Conection
Diagram

| BIT | JM1 TO JM7 |
|---|---|
| 0 | A |
| 1 | B |
| 2 | C |
| 3 | D |

## 5 or 6 wires motor



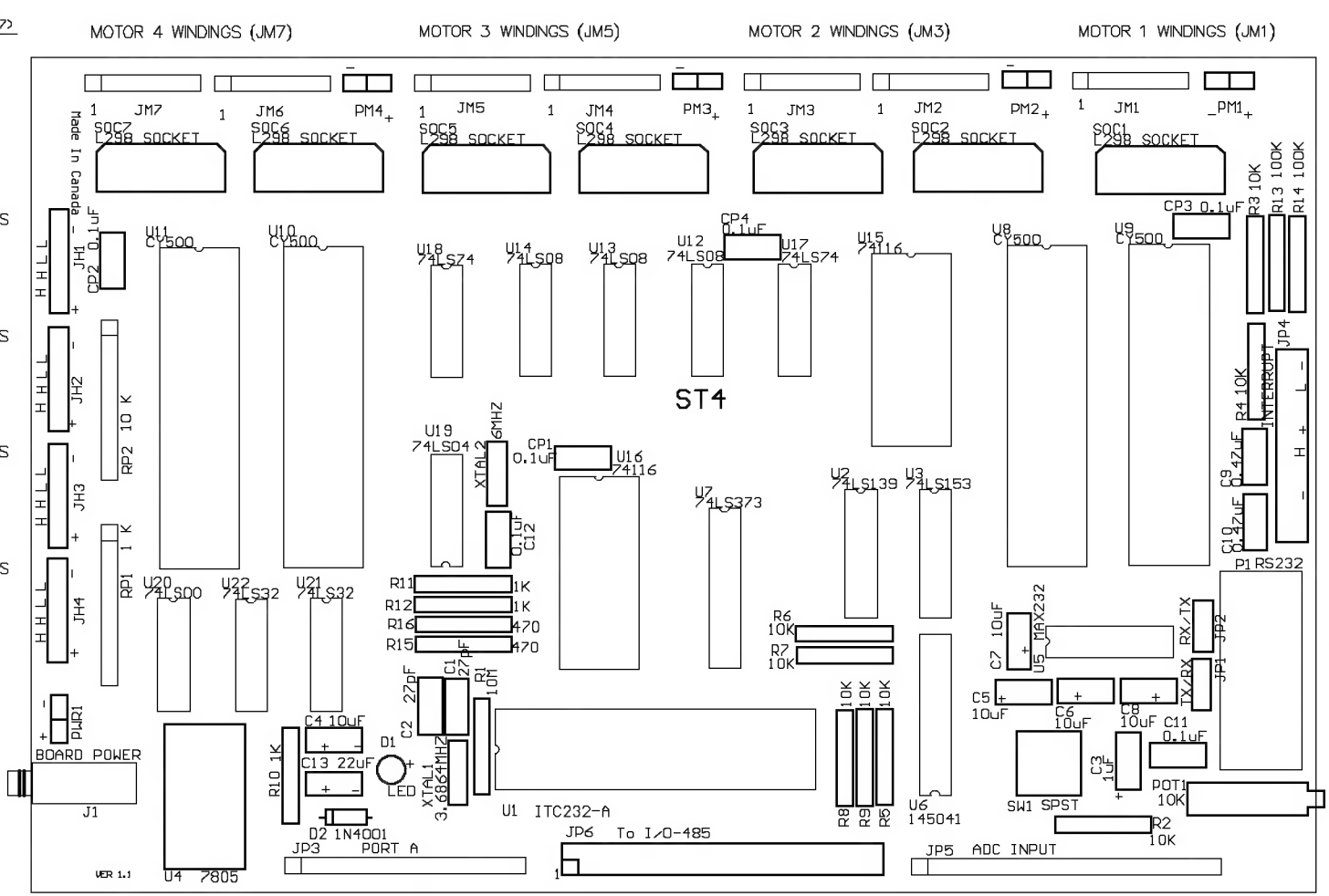

All diodes Shoktty or Fast Recovery rated at motor voltage and current.

# STEPPER MOTOR BOARD SERIES – CONNECTION DIAGRAM

Reffer to the motor data sheets
for identifying coil phases.

Motor windings phase
(connectors JM1 to JM7)

1- A
2- B
3- C
4- C




LIMIT SWITCH INPUTS
MOTOR 1 (JH1)

LIMIT SWITCH INPUTS
MOTOR 2 (JH2)

LIMIT SWITCH INPUTS
MOTOR 3 (JH3)

LIMIT SWITCH INPUTS
MOTOR 4 (JH4)

IRQ INPUTS (JP4)

RS232
SERIAL CONNECTOR (P1)

BOARD POWER INPUTS
(J1 OR PWR1)
(7.5 to 12 VDC @ 1Amp)

DIGITAL I/O CONNECTOR (JH3)  NETWORK CARD CONNECTOR (JP6)  ANALOG INPUTS CONNECTOR (JP5)